```
%
%%%%%%%  small_network_mcmc.m   %%%%%%%
%
%
% This program utilizes a Markov Chain Monte Carlo
% method for the prediction of the most likely configuration
% or state of a regulatory network.
%
%
% smg - Last Modified: 8/7/00

frame_count = 0;
rejected = 0;
edge_number(1,1) = 0;

% Ask user for set-up information
configuration = input ('Starting from scratch (0) or from an initial Adjacency mtx (1):
');
if configuration == 1
   A_dir = input ('Enter path and filename for Adjacency matrix:');
else
   initial_edges = input('Input initial number of edges: ');
end
input_dir = input('Enter directory for probability matrices: ');
iterations = input('Input number of iterations: ');
burn_in = input('Enter number of burn-in iterations: ');
directory = input('Enter directory of input files in single quotes: ');
ks = input('Enter maximum number of edges to sample(change) at one time: ');
save_interval = input('Enter save interval: ');


% Load precalculated arrays
cd(input_dir);
%load edge_matrix;               % used for sampling edges
load edge_probabilities;
probarray = edge_probabilities;  % probability of an edge
clear edge_probabilities;
load edge_probabilities_zero
probarray_zero = edge_probabilities_zero; % probability of not having edge
clear edge_probabilities_zero;


% Load look-up table for ln of factorial
load lnfac.txt -ascii;
lnfac = reshape(lnfac',10008,1);




% Load data
cd(directory);
names = dir;

% get vertices
for n = 3:length(names);       %3-708(Curagen) 3-13(test) 3-641(DIP) - process each file
   filename = names(n).name;
   datain = load(filename);
   protein(n-2).id = n-2;
   newname = strtok(filename,'.');
   newname = str2num(newname);
   protein(n-2).name = newname;
   protein(n-2).domains = datain(1:end);
   idnamelist(n-2) = newname;
```

```
end
len = length(idnamelist);          % number of vertices

edge_freq = sparse(len,len);

% Set up random edges between vertices
if configuration == 0
   A = sparse(len,len);
   for i = 1:initial_edges
      i_init = 1 + floor(rand*len);
      j_init = 1 + floor(rand*len);
      A(i_init,j_init) = 1;
   end
else
   % must be called Alast - Make more user friendly
   load(A_dir);
   A = Alast;  % A = Alast;
   clear Alast;
end


edgesinit = nnz(A)          % number of interactions
edgevector = sum(A,2);      % number of outgoing edges for each vertex
inedgevector = sum(A);      % number of incoming edges for each vertex
edge_number(1,1) = edgesinit;

% Calculate edge probabilities
% prob_edges is the sum of log(prob) for each existing edge
% prob_edges_zero is the sum of log(prob) for each non-existant edge
prob_edges = 0;
prob_edges_zero = 0;
for i = 1:len
   for j = 1:len
      if A(i,j) == 1
         prob_edges = prob_edges + log(probarray(i,j));
      else
         prob_edges_zero = prob_edges_zero + log(probarray_zero(i,j));
      end
   end
end


%%%%%%%%%%%%%%%% Calculate system probability %%%%%%%%%%%%%%%
LOGPTRANS = prob_edges;
LOGPTRANSzero = prob_edges_zero;




%%%%%%%%%%%%%%%%% Incoming edge probability - Multinomial
% Determine the number of vertices with 0,1,.. incoming edges
invertexedgevect = zeros(1,max(inedgevector)+1);
for i = 1:length(inedgevector)         % i = 1 is vert w/ zero incoming edges
   invertexedgevect(inedgevector(i)+1) = invertexedgevect(inedgevector(i)+1) + 1;
end

% Multinomial dist.
MIN = sum(invertexedgevect);      % Total number of vertices
LOGPIN = 0;
LOGDENIN = 0;
for i = 1:length(invertexedgevect)
   LOGPIN = LOGPIN + invertexedgevect(i)*log(probinedge(i-1));
   LOGDENIN = LOGDENIN + lnfac(invertexedgevect(i)+ 1);
end
LOGNUMIN = lnfac(MIN+1);
```

```
%%%%%%%%%%%%%%%%%%%% Outgoing edge probability - Multinomial
% Determine the number of vertices with 0,1,.. outgoing edges
vertexedgevect = zeros(1,max(edgevector)+1);
for i = 1:length(edgevector)         % i = 1 is vertex w/ 0 outgoing edges
   vertexedgevect(edgevector(i)+1) = vertexedgevect(edgevector(i)+1) + 1;
end

% Multinomial dist.
M = sum(vertexedgevect);           % Total number of vertices (= MIN)
LOGPOUT = 0;
LOGDEN = 0;
for i = 1:length(vertexedgevect)
   LOGPOUT = LOGPOUT + vertexedgevect(i)*log(edgedist(i-1));
   LOGDEN = LOGDEN + lnfac(vertexedgevect(i)+ 1);
end
LOGNUM = lnfac(M+1);
LOGDEN;
LOGPOUT;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Likelihood of system in initial configuration
LOGSYSX = (LOGNUM-LOGDEN)+LOGPOUT+(LOGNUMIN-LOGDENIN)+LOGPIN+LOGPTRANS+LOGPTRANSzero;
fprintf('Initial Probability = %5.4E\n\n',LOGSYSX)
%

% Save initial configuration - for reference
LOGSYSXorig = LOGSYSX;
Aorig = A;
edgevectororig = edgevector;
inedgevectororig = inedgevector;
vertexedgevectorig = vertexedgevect;
invertexedgevectorig = invertexedgevect;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Add or remove edge at random & recompute likelihood

flag = 1;             % for testing purposes
if flag == 1;

prob_edgescomp = prob_edges;
prob_edges_zerocomp = prob_edges_zero;
edgevectorcomp = edgevector;
inedgevectorcomp = inedgevector;
Acomp = A;


% set up matrix for observing frequency of edge sampling
edge_freq = zeros(len,len);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic;
for x = 1:iterations
   % choose to add or remove
   if rand < 0.5
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      % add edge
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      % create A0 matrix
      A0 = (A==0);
```

```
    % select edge from those that are currently = 0
    [edges,q_x2y,q_y2x] = edge_selection(ks,A0,lnfac);

    [edge_rows,edge_cols] = size(edges);
    % update parameters
    for t = 1:edge_rows
       A(edges(t,1),edges(t,2)) = 1;
       prob_edges = prob_edges + log(probarray(edges(t,1),edges(t,2)));
       prob_edges_zero = prob_edges_zero - log(probarray_zero(edges(t,1),edges(t,2)));
       edgevector(edges(t,1)) = edgevector(edges(t,1)) + 1;     % update edge vectors
       inedgevector(edges(t,2)) = inedgevector(edges(t,2)) + 1;
       edge_freq(edges(t,1),edges(t,2)) = edge_freq(edges(t,1),edges(t,2)) + 1;
    end


    % flag showing state
    add_edge = 1;

 else
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % remove an edge
    % select preferentially from proteins that should be less well connected
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % select edge to remove
    [edges,q_x2y,q_y2x] = edge_selection(ks,A,lnfac);


    % change edge
    [edge_rows,edge_cols] = size(edges);

    % update parameters
    for t = 1:edge_rows
       A(edges(t,1),edges(t,2)) = 0;
       prob_edges = prob_edges - log(probarray(edges(t,1),edges(t,2)));
       prob_edges_zero = prob_edges_zero + log(probarray_zero(edges(t,1),edges(t,2)));
       edgevector(edges(t,1)) = edgevector(edges(t,1)) - 1;
       inedgevector(edges(t,2)) = inedgevector(edges(t,2)) - 1;
       edge_freq(edges(t,1),edges(t,2)) = edge_freq(edges(t,1),edges(t,2)) + 1;
    end



    % flag showing state
    add_edge = 0;

 end


 %%%%%%%%%%%
 % Essentially, a repeat of initial system probability calculation

 LOGPTRANSY = prob_edges;              % sum of transition probabilities for
 LOGPTRANSzeroY = prob_edges_zero;     % new system (w/ or w/o new edge)


 %%%%%%%%%%%%%%%%%%%%%%%%%%
 % Incoming edge probability
 invertedgenew = zeros(1,max(inedgevector)+1);   % # of vert. w/ 0,1,...edges
 for i = 1:length(inedgevector)              % i = 1 -> vert w/ 0 incoming edges
    invertedgenew(inedgevector(i)+1) = invertedgenew(inedgevector(i)+1) + 1;
 end

 MINY = sum(invertedgenew);     % Calc Multinomial
 LOGPINY = 0;
```

```
LOGDENINY = 0;
for i = 1:length(invertedgenew)
   LOGPINY = LOGPINY + invertedgenew(i)*log(probinedge(i-1));
   LOGDENINY = LOGDENINY + lnfac(invertedgenew(i)+ 1);
end
LOGNUMINY = lnfac(MINY+1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outgoing edge probability
vertedgenew = zeros(1,max(edgevector)+1);
for i = 1:length(edgevector)
   vertedgenew(edgevector(i)+1) = vertedgenew(edgevector(i)+1) + 1;
end

MY = sum(vertedgenew);      % Calc Multinomial
LOGPOUTY = 0;
LOGDENY = 0;
for i = 1:length(vertedgenew)
   LOGPOUTY = LOGPOUTY + vertedgenew(i)*log(edgedist(i-1));
   LOGDENY = LOGDENY + lnfac(vertedgenew(i)+ 1);
end
LOGNUMY = lnfac(MY+1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probability of new configuration
LOGSYSY = (LOGNUMY-LOGDENY)+LOGPOUTY+(LOGNUMINY-
LOGDENINY)+LOGPINY+LOGPTRANSY+LOGPTRANSzeroY;


% Check if iterations>burn_in & establish plotting interval based on
% the number of edges existing after burn in
if x == burn_in
   %save_interval = nnz(A);
   %Astart = A;
   Asave = A;
   saves = 1;
   SYSVECT(saves,1) = x;
   SYSVECT(saves,2) = LOGSYSX;
   edge_number(1,saves)=nnz(A);
   frame_count = 1;
   frame(frame_count).adj = Asave;
   frame(frame_count).edge_num = nnz(A);
end



% Save configuration after every 'save_interval' iterations
if x > burn_in & rem(x,save_interval) == 0
   Asave = Asave + A;
   saves = saves + 1;
   SYSVECT(saves,1) = x;
   SYSVECT(saves,2) = LOGSYSX;
   edge_number(1,saves) = nnz(A);
end


% Save "frames" for animation of edge probabilities
if x > burn_in & rem(x,200*save_interval) == 0
   frame_count = frame_count + 1;
   frame(frame_count).adj = Asave;
   frame(frame_count).edge_num = nnz(A);
end
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % probability of accepting edge
    paccept = exp((LOGSYSY + q_y2x) - (LOGSYSX + q_x2y));
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    if rand <= paccept            % decide whether or not to keep change
       % keep change
       Acomp = A;
       edgevectorcomp = edgevector;
       inedgevectorcomp = inedgevector;
       prob_edgescomp = prob_edges;
       prob_edges_zerocomp = prob_edges_zero;

       fprintf('%5.0d\tpaccept=%3.2E\tEdges=%4d\nLOGOLD=%5.4E\tLOGNEW=%5.4E\tDiff=%5.4E\n',
x,paccept,nnz(A),LOGSYSX,LOGSYSY,LOGSYSY-LOGSYSX)
       fprintf('q_x2y= %5.4E\tq_y2x= %5.4E\n\n',q_x2y,q_y2x)

       LOGSYSX = LOGSYSY;


       % update sampling distributions appropriately when an edge is added
       % or removed

    else
       % proposed addition or removal of edge is not accepted
       rejected = rejected + 1;
       A = Acomp;                  % reset values to that of the previous iteration
       edgevector = edgevectorcomp;
       inedgevector = inedgevectorcomp;
       prob_edges = prob_edgescomp;
       prob_edges_zero = prob_edges_zerocomp;
    end
end
end
toc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%save d:\apps\matlabr11\work\dipdata\Astart.mat Astart;
%save d:\apps\matlabr11\work\dipdata\Asave.mat Asave;

rejected/iterations                % percent of changes rejected
nnz(A)
%figure;
%gplot(Abest,nodeloc,'-o');

clear Acomp;
clear A0;


figure;
plot(SYSVECT(:,1),SYSVECT(:,2));    % Plot system probability at each 'save_interval'
xlabel('Iteration Number')
ylabel('Log(L)')
title(['MCMC test - ',date])
%text(14000,-1608,'1500 edges-start, ### edges-end, s = 0.025, 50% Rejection')

figure;
pcolor(Asave./saves);
shading interp
colorbar('v')
xlabel('Vertex')
```

```
ylabel('Vertex')
title('Fraction of Time Edge is Occupied')

figure;
plot(edge_number)
xlabel('Save Number')
ylabel('Number of Edges')

figure;
pcolor(edge_freq);
shading interp
colorbar('v')
title('Frequency of Edge Sampling');
```

```
function [out,q_x2y,q_y2x] = edge_selection(ks,A,lnfac)
%
%      out = edge_selection(ks,A,lnfac)
%
%
%   This program is used for randomly selecting edges
%   from an adjacency matrix.
%
%   Input A is an adjacency (or inverse adjacency) matrix.
%   'ks' is the maximum number of edges to be sampled.
%


len = length(A);
total_edges = len*len;

% find all 1's in A
[source,target] = find(A==1);

% number of interactions available for sampling
edges_avail = length(source);


% number of edges to be sampled
edges_sampled = 1 + floor(rand*ks);     %rand*edges_avail


% make sure you don't sample from more than available
if edges_sampled > edges_avail
   edges_sampled = edges_avail;      %useful only for ks case
end


% mix edges to be sampled
choose_from_this = randperm(edges_avail);

% Sample edges from 1 to edges_sampled
out = zeros(edges_sampled,2);    %initialize out

out(:,1) = source(choose_from_this(1:edges_sampled));
out(:,2) = target(choose_from_this(1:edges_sampled));


% Calculate the proposal distribution from new to old and old to new
qx = 0;
qy = 0;
opp_edges = total_edges - edges_avail + edges_sampled;

for i = 1:edges_sampled
   qx = qx + log((edges_sampled - i + 1)/(edges_avail - i + 1));
   qy = qy + log((edges_sampled - i + 1)/(opp_edges - i +1));
end
q_x2y = qx;

q_y2x = qy;
```

```
function [out,q_x2y,q_y2x] = edge_selection_old(ks,A,lnfac)
%
%     out = edge_selection_old(ks,A,lnfac)
%
%
%  This program is used for randomly selecting edges
%  from an adjacency matrix.
%
%  Input A is an adjacency (or inverse adjacency) matrix.
%  'ks' is the maximum number of edges to be sampled.
%


len = length(A);
total_edges = len*len;

% find all 1's in A
[source,target] = find(A==1);

% number of interactions available for sampling
edges_avail = length(source);


% number of edges to be sampled
edges_sampled = 1 + floor(rand*ks);    %rand*edges_avail


% make sure you don't sample from more than available
if edges_sampled > edges_avail
   edges_sampled = edges_avail;     %useful only for ks case
end


% sample interactions available and place x and y coordinates in "out"
out = zeros(edges_sampled,2);    %initialize out
[rows,cols] = size(out);
i = 1;
while i <= edges_sampled
   x = 1 + floor(rand*edges_avail);        % pick vertex
   for n = 1:rows
      if intersect([source(x),target(x)],out,'rows')
         % sampling previously sampled point
         % try another
         break;
      else
         % previously unsampled - place in out matrix
         out(i,1) = source(x);
         out(i,2) = target(x);
         i = i + 1;
      end
   end
end
out;


% Calculate the proposal distribution from new to old
qx = 0;
qy = 0;
opp_edges = total_edges - edges_avail + edges_sampled;

for i = 1:edges_sampled
   qx = qx + log((edges_sampled - i + 1)/(edges_avail - i + 1));
   qy = qy + log((edges_sampled - i + 1)/(opp_edges - i +1));
end
q_x2y = qx;
q_y2x = qy;
```

```
function [out,q_x2y,q_y2x] = edge_selection(ks,A,lnfac)
%
%     out = edge_selection(ks,A,lnfac)
%
%
%  This program is used for randomly selecting edges
%  from an adjacency matrix.
%
%  Input A is an adjacency (or inverse adjacency) matrix.
%  'ks' is the maximum number of edges to be sampled.
%


len = length(A);
total_edges = len*len;

% find all 1's in A
[source,target] = find(A==1);

% number of interactions available for sampling
edges_avail = length(source);


% number of edges to be sampled
edges_sampled = 1 + floor(rand*ks);    %rand*edges_avail


% make sure you don't sample from more than available
if edges_sampled > edges_avail
   edges_sampled = edges_avail;     %useful only for ks case
end


% mix edges to be sampled
choose_from_this = randperm(edges_avail);

% Sample edges from 1 to edges_sampled
out = zeros(edges_sampled,2);    %initialize out

out(:,1) = source(choose_from_this(1:edges_sampled));
out(:,2) = target(choose_from_this(1:edges_sampled));


% Calculate the proposal distribution from new to old and old to new
qx = 0;
qy = 0;
opp_edges = total_edges - edges_avail + edges_sampled;

for i = 1:edges_sampled
   qx = qx + log((edges_sampled - i + 1)/(edges_avail - i + 1));
   qy = qy + log((edges_sampled - i + 1)/(opp_edges - i +1));
end
q_x2y = qx;

q_y2x = qy;
```

```
function [out,q_x2y,q_y2x] = edge_selection_old(ks,A,lnfac)
%
%     out = edge_selection_old(ks,A,lnfac)
%
%
%  This program is used for randomly selecting edges
%  from an adjacency matrix.
%
%  Input A is an adjacency (or inverse adjacency) matrix.
%  'ks' is the maximum number of edges to be sampled.
%


len = length(A);
total_edges = len*len;

% find all 1's in A
[source,target] = find(A==1);

% number of interactions available for sampling
edges_avail = length(source);


% number of edges to be sampled
edges_sampled = 1 + floor(rand*ks);    %rand*edges_avail


% make sure you don't sample from more than available
if edges_sampled > edges_avail
   edges_sampled = edges_avail;     %useful only for ks case
end


% sample interactions available and place x and y coordinates in "out"
out = zeros(edges_sampled,2);    %initialize out
[rows,cols] = size(out);
i = 1;
while i <= edges_sampled
   x = 1 + floor(rand*edges_avail);        % pick vertex
   for n = 1:rows
      if intersect([source(x),target(x)],out,'rows')
         % sampling previously sampled point
         % try another
         break;
      else
         % previously unsampled - place in out matrix
         out(i,1) = source(x);
         out(i,2) = target(x);
         i = i + 1;
      end
   end
end
out;


% Calculate the proposal distribution from new to old
qx = 0;
qy = 0;
opp_edges = total_edges - edges_avail + edges_sampled;

for i = 1:edges_sampled
   qx = qx + log((edges_sampled - i + 1)/(edges_avail - i + 1));
   qy = qy + log((edges_sampled - i + 1)/(opp_edges - i +1));
end
q_x2y = qx;
q_y2x = qy;
```

```
function prob = edgedist(numedges)
%
%  This function takes as input the number of outgoing
%  edges from a vertex and returns the probability of
%  this as "prob"
%
%  Based on fits to experimental data
%
% smg - Last Modified: 7/22/00
%

if numedges == 0
   prob = 0.489751;
else
   prob = 0.304647*numedges.^(-1.9691);
end
```

```
function y = probinedge(numedges)
%
%
%
%
%


if numedges == 0
   y = 0.306735;
else
   y = 0.556352*numedges.^(-2.8037);
end
```

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)__________ of ______________________________ were not present for scanning. (Document title)

☐ Page(s)__________ of ______________________________ were not present for scanning. (Document title)

▫ ***Scanned copy is best available.*** Drawings: Figure on the Sheet # 4 of 12 is very dark